INFORMATICĂ · CLASA A X-A · TABLOURI UNIDIMENSIONALE · LECȚIA 7

Secvențe în vectori, sume parțiale și secvența de sumă maximă

Durată: 60 min Nivel: toate profilurile Înainte: lecțiile x-02

CE VEI ȘTI DUPĂ LECȚIE

- Înțelegi diferența dintre secvență (elemente consecutive) și subșir.
- Determini cea mai lungă secvență cu o proprietate, într-o parcurgere.
- Calculezi suma oricărei secvențe în O(1) cu sume parțiale.
- Găsești secvența de sumă maximă (algoritmul lui Kadane).

1. Secvență vs. subșir

- **Secvență** = elemente **consecutive**: `a[i], a[i+1], …, a[j]`.
- **Subșir** = elemente în aceeași ordine, dar nu neapărat vecine.

Într-un vector cu n elemente există `n(n+1)/2` secvențe.

DE CE AȘA?

De ce n(n+1)/2? O secvență e determinată de capetele (i, j) cu i ≤ j. Pentru i = 1 sunt n variante pentru j, pentru i = 2 sunt n − 1, ..., pentru i = n una singură: suma 1 + 2 + ... + n. De aceea o soluție care încearcă **toate** secvențele face $O(n^2)$ pași, prea mult pentru $n = 10^6$.

2. Cea mai lungă secvență cu o proprietate

Același tipar ca la clasa a IX-a, acum cu poziții (ca să putem afișa capetele):

```cpp
#include <iostream>
using namespace std;

int a[1001], n;

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    int lung = 0, lmax = 0, stMax = 0;
    for (int i = 1; i <= n; i++)
    {
        if (a[i] % 2 == 0) lung++;    // elementul continuă secvența
        else lung = 0;                // secvența se rupe
        if (lung > lmax) { lmax = lung; stMax = i - lung + 1; }
    }
    cout << stMax << ' ' << lmax;     // secvența: a[stMax .. stMax + lmax - 1]
    return 0;
}
```

Pentru proprietăți care depind de vecin (crescătoare, elemente egale), compară `a[i]` cu `a[i-1]`.

i	a[i]	par?	lung	lmax	stMax
1	3	nu	0	0	0
2	4	da	1	1	2
3	6	da	2	2	2
4	8	da	3	3	2
5	5	nu	0	3	2
6	2	da	1	3	2
7	4	da	2	3	2

Programul afișează `2 3`: cea mai lungă secvență de numere pare începe la poziția 2 și are 3 elemente (4, 6, 8).

DE CE AȘA?

De ce `stMax = i - lung + 1`? Dacă la poziția `i` secvența curentă are `lung` elemente, ea începe cu `lung − 1` poziții mai în urmă. Exemplu: la i = 4, lung = 3: începe la 4 − 3 + 1 = 2.

3. Sume parțiale

Problema: avem un vector și **multe întrebări** de forma „care e suma elementelor de la poziția x la poziția y?". Dacă pentru fiecare întrebare adunăm, facem O(n) pe întrebare, prea lent pentru 10^5 întrebări.

Ideea: precalculăm `s[i] = a[1] + a[2] + … + a[i]`.



Sume parțiale

```cpp
#include <iostream>
using namespace std;

int a[100001], n, m;
long long s[100001];

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {
        cin >> a[i];
        s[i] = s[i - 1] + a[i];        // s[0] = 0 (global)
    }
    for (int q = 1; q <= m; q++)
    {
        int x, y;
        cin >> x >> y;
        cout << s[y] - s[x - 1] << '\n';    // suma a[x..y]
    }
    return 0;
}
```

i	1	2	3	4	5
a[i]	3	1	4	1	5
s[i]	3	4	8	9	14

Suma `a[2..4]` = `s[4] − s[1]` = 9 − 3 = **6** (= 1 + 4 + 1 ✓).

DE CE AȘA?

De ce `s[y] − s[x − 1]`? `s[y]` conține tot de la început până la y; `s[x − 1]` conține partea de dinaintea lui x. Diferența păstrează exact `a[x..y]`. De aceea `s[0] = 0` e necesar: pentru x = 1 se scade 0. Precalcularea costă O(n), fiecare întrebare O(1): total O(n + m).

SFAT

Sumele parțiale merg pe orice „numărare cumulată”: câte numere prime sunt până la x (combinat cu ciurul), câte elemente pare sunt în a[x..y] etc.

4. Secvența de sumă maximă (Kadane)

Vector cu numere întregi (și negative). Ce secvență are suma cea mai mare?

Ideea: parcurgem vectorul și ținem suma secvenței curente. Dacă suma curentă devine negativă, nu are rost să o continuăm: o luăm de la capăt.



când sc devine negativ, secvența începe din nou de la elementul curent

Kadane

```cpp
#include <iostream>
using namespace std;

int a[1001], n;

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    long long sc = 0, smax = a[1];
    int st = 1, bestSt = 1, bestDr = 1;
    for (int i = 1; i <= n; i++)
    {
        if (sc < 0) { sc = 0; st = i; }    // începem o secvență nouă la i
        sc += a[i];
        if (sc > smax) { smax = sc; bestSt = st; bestDr = i; }
    }
    cout << smax << ' ' << bestSt << ' ' << bestDr;
    return 0;
}
```

Complexitate **O(n)**. (Soluția cu toate perechile `i, j` ar fi O(n²) sau chiar O(n³).)

i	a[i]	sc înainte	sc după	smax	secvența
1	2	0	2	2	[1, 1]
2	−5	2	−3	2	[1, 1]
3	3	−3 → 0 (reset)	3	3	[3, 3]
4	4	3	7	7	[3, 4]
5	−1	7	6	7	[3, 4]
6	2	6	8	8	[3, 6]
7	−6	8	2	8	[3, 6]
8	1	2	3	8	[3, 6]

DE CE AȘA?

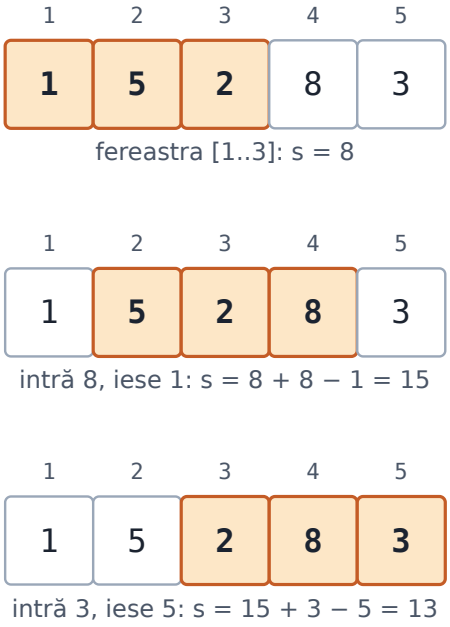
De ce se resetează când `sc` e negativ? O sumă negativă doar **scade** orice secvență care ar continua-o; pornind de la zero, orice continuare e măcar la fel de bună. De aceea nu ratăm niciodată secvența optimă. (Iar `smax = a[1]` și nu 0: dacă toate elementele sunt negative, răspunsul este cel mai mare element, nu „secvența vidă”.)

LA BAC

Kadane este un exemplu perfect de algoritm „din mers” pentru III.3: o singură parcurgere, câteva variabile, deci funcționează și citind dintr-un fișier cu 10^6 numere.

5. Fereastra glisantă (secvențe de lungime fixă k)

Suma maximă a unei secvențe de exact k elemente:



Fereastra glisantă

```cpp
#include <iostream>
using namespace std;

int a[100001], n, k;

int main()
{
    cin >> n >> k;
    for (int i = 1; i <= n; i++) cin >> a[i];
    long long s = 0;
    for (int i = 1; i <= k; i++) s += a[i];
    long long best = s;
    for (int i = k + 1; i <= n; i++)
    {
        s += a[i] - a[i - k];     // intră a[i], iese a[i-k]
        if (s > best) best = s;
    }
    cout << best;
    return 0;
}
```

DE CE AȘA?

De ce nu recalculăm suma ferestrei de la zero? O fereastră nouă diferă de cea veche prin **un singur element care intră și unul care iese**: ajustăm suma în O(1). Recalcularea ar costa O(k) pe poziție, adică O(n·k) în total.

6. Greșeli frecvente

Greșeala	Efect	Corect
`s[x] - s[y - 1]` (ordine inversă)	suma cu semn greșit	`s[y] - s[x - 1]`
uiți `s[0] = 0` / `x - 1` negativ	rezultat greșit pentru `x = 1`	`s[0] = 0`
`int` pentru sume mari	depășire	`long long`
`smax = 0` la Kadane	greșit dacă toate sunt negative	`smax = a[1]`
`lung = 1` în loc de `0` la ruperea unei secvențe de pare	numeri un impar	`lung = 0`

7. La BAC

LA BAC

„Cea mai lungă secvență” (III.3) se rezolvă cu tiparul din secțiunea 2, o parcurgere și memorie constantă. Sumele parțiale și Kadane apar mai ales la concursuri, dar ideea de „precalculez ca să răspund repede” e folosită în orice problemă cu multe întrebări.

8. Exersează pe hârtie

Răspunsurile sunt la final: încearcă întâi singur.

1. Vectorul `(2, -5, 3, 4, -1, 2, -6, 1)`. Care e secvența de sumă maximă? Urmărește Kadane pas cu pas.

2. Câte secvențe are un vector cu 6 elemente?

3. Se dă un vector de 0 și 1 și m întrebări (x, y): câți de 1 sunt între pozițiile x și y?

4. Pentru `a = (3, 1, 4, 1, 5)`, cât este suma `a[3..5]` folosind sumele parțiale?

5. Pentru `a = (1, 5, 2, 8, 3)` și `k = 3`, care e suma maximă a unei ferestre de lungime 3?

6. Ce afișează Kadane (valoarea `smax`) pentru `(-3, -1, -2)`? Ce s-ar întâmpla cu `smax = 0`?

7. De ce soluția cu două bucle (toate perechile `i, j`) nu merge pentru n = 10^6?

9. Mai departe (pentru cine vrea mai mult)

Sume parțiale 2D. Pe o matrice poți precalcula `s[i][j]` = suma dreptunghiului din colțul (1,1) până la (i,j); apoi suma oricărui dreptunghi se află din 4 valori (principiul includerii–excluderii). Folosit la procesarea imaginilor („integral images").

Diferențe (partea inversă). Pentru **adunări pe secvențe** („adaugă 5 la a[x..y]", de multe ori) folosești vectorul de diferențe `d[x] += 5; d[y+1] -= 5;` și la final calculezi sume parțiale. Rezolvi m actualizări în O(n + m).

Două capete (two pointers). Pentru „cea mai lungă secvență cu suma ≤ S" (valori pozitive) folosești două capete care se mișcă doar în față: O(n). Este o variantă a ferestrei glisante cu lungime variabilă.

Kadane în viața reală. Algoritmul a fost inventat de Jay Kadane (1984) pentru analiza de imagini, și azi stă la baza unor analize din finanțe (perioada cu cea mai mare câștig consecutiv) și bioinformatică (regiuni de ADN cu scor maxim).

Rezumat

- Secvență = elemente consecutive; există n(n+1)/2.
- Cea mai lungă secvență cu P: crește lungimea sau resetează.
- Sume parțiale: `s[i] = s[i-1] + a[i]`; suma `a[x..y] = s[y] − s[x−1]`, în O(1).
- Kadane: resetezi suma curentă când devine negativă; `smax = a[1]`.
- Fereastră glisantă: `s += a[i] − a[i−k]`.

Probleme de lucru (pbinfo)

Începe cu acestea, în ordine: sunt alese să urmeze pașii din lecție.

- [ ] **#516** **sumsec1** — ușoară
  https://www.pbinfo.ro/probleme/516/sumsec1

- [ ] #523 **secvegale** — ușoară
  https://www.pbinfo.ro/probleme/523/secvegale
- [ ] #299 **sumesecv** — ușoară
  m întrebări de sumă → sume parțiale.
  https://www.pbinfo.ro/probleme/299/sumesecv
- [ ] #297 **secvsummax** — medie
  https://www.pbinfo.ro/probleme/297/secvsummax

Temă (pbinfo)

Trimite soluțiile pe pbinfo și notează punctajul obținut. Bifează ce ai terminat.

- [ ] #518 **secvzero** — ușoară
  https://www.pbinfo.ro/probleme/518/secvzero
- [ ] #166 **secvmax** — medie
  https://www.pbinfo.ro/probleme/166/secvmax
- [ ] #181 **secvcresc** — medie
  https://www.pbinfo.ro/probleme/181/secvcresc
- [ ] #304 **secvente** — ușoară
  https://www.pbinfo.ro/probleme/304/secvente
- [ ] #2921 **sumesecv1** — medie
  https://www.pbinfo.ro/probleme/2921/sumesecv1
- [ ] #300 **sumainsecv** — ușoară
  https://www.pbinfo.ro/probleme/300/sumainsecv
- [ ] #4478 **nr-max-de-zerouri** — ușoară
  https://www.pbinfo.ro/probleme/4478/nr-max-de-zerouri

Pentru cei care vor mai mult

Probleme mai grele sau de concurs, pentru cine vrea să meargă dincolo de BAC.

- [ ] #524 **secvegale1** — medie
  https://www.pbinfo.ro/probleme/524/secvegale1
- [ ] #2142 **easy-sum** — medie
  https://www.pbinfo.ro/probleme/2142/easy-sum

Grile din subiectele de BAC (pbinfo)

Exerciții de tip grilă din variantele de Bacalaureat, rezolvate online pe pbinfo, cu verificare automată a răspunsului.

- [ ] #1035 **Exercițiul #1035**
  suma elementelor după o condiție, pe un vector dat
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1035
- [ ] #1040 **Exercițiul #1040**
  prima și ultima poziție unde apare un element
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1040
- [ ] #1036 **Exercițiul #1036**
  care secvență afișează corect produsul elementelor pare
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1036
- [ ] #1037 **Exercițiul #1037**
  a[i] = i·i − i: câte componente satisfac o condiție
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1037

Citește și exersează în plus

- Secvențe în vectori https://www.pbinfo.ro/articole/5613/secvente-in-vectori
- Sume parțiale în tablouri https://www.pbinfo.ro/articole/5615/sume-partiale-in-tablouri
- Secvență de sumă maximă https://www.pbinfo.ro/articole/20499/secventa-de-suma-maxima
- CSES — Maximum Subarray Sum — exact algoritmul lui Kadane, cu teste mari (engleză)
  https://cses.fi/problemset/task/1643
- VisuAlgo — Maximum Subarray / Sliding Window — caută animații pentru sume parțiale și ferestre https://visualgo.net/en
- Python Tutor pentru C++ — urmărește sc și smax pas cu pas https://pythontutor.com/cpp.html

Răspunsuri

Exersează pe hârtie

1. Suma maximă este **8**, pe secvența `a[3..6] = (3, 4, −1, 2)` (3 + 4 − 1 + 2 = 8). La `i = 3`, `sc = −3` e negativ, deci se resetează la 0 și secvența începe la 3.

2. 21 de secvențe: 6 · 7 / 2.

3. Se construiește `s[i]` = numărul de 1 de la început până la `i` (`s[i] = s[i-1] + a[i]`), iar răspunsul este `s[y] − s[x − 1]` : aceeași metodă ca la sume, pentru că suma elementelor unui vector de 0 și 1 numără exact 1-urile.

4. `s = (3, 4, 8, 9, 14)`, deci `a[3..5] = s[5] − s[2] = 14 − 4 = 10` (4 + 1 + 5).

5. `15`. Ferestrele: 1+5+2 = 8; 5+2+8 = 15; 2+8+3 = 13.

6. `smax = −1` (cel mai mare element). Cu `smax = 0` programul ar afișa 0, adică o sumă care nu aparține niciunei secvențe nevide: greșit.

7. Pentru n = 10^6 sunt aproximativ $5 \cdot 10^{11}$ perechi, deci timpul ar fi de ordinul orelor (limita este ~10^8 operații pe secundă). Kadane face doar 10^6 pași.